The 5th International Symposium on Virtual Reality, Archaeology, and Cultural Heritage VAST (2004)
K. Cain, Y. Chrysanthou, F. Niccolucci, N. Silberman (Editors)

Visualizing Design and Spatial Structure of Ancient Architecture using Blueprint Rendering

M. Nienhaus and J. Döllner

University of Potsdam, Hasso Plattner Institute, Germany

Abstract

We present the blueprint rendering technique as an effective tool for interactively visualizing, exploring, and communicating the design and spatial structure of ancient architecture by outlining and enhancing their visible and occluded features. The term blueprint in its original meaning denotes "a photographic print in white on a bright blue ground or blue on a white ground used especially for copying maps, mechanical drawings, and architects' plans" (Merriam Webster). Blueprints consist of transparently rendered features, represented by their outlines. This way, blueprints allow for realizing complex, hierarchical object assemblies such as architectural drafts. Our technique renders 3D models of architecture to automatically generate blueprints that provide spatial insights, and generates plan views that provide a systematic overview, and enhances these drafts using glyphs. Additionally, blueprint rendering can highlight features of particular importance and their relation to the entire structure, and can reduce visual complexity if the structural complexity of the 3D model is excessive.

Categories and Subject Descriptors (according to ACM CCS): I.3.3 [Computer Graphics]: Applications

1. Introduction

This paper describes the novel and innovative blueprint rendering technique applied to architecture of cultural heritage to effectively visualize and communicate their design and spatial structure. It represents a real-time non-photorealistic rendering technique that enhances visually important edges of visible and occluded features of 3D models, e.g., architecture models. In contrast to a wire-frame depiction, which complicates the visual perception of complex object assemblies because it does not differentiate between triangulation edges and actual outlines (e.g., silhouettes), and transparency renderings, whose outlines are hardly noticeable, in particular in regions of high depth complexity, the blueprint rendering technique generates vivid and expressive depictions that facilitate visual perception (Fig. 1). Furthermore, its resulting depictions can be combined with further 3D scene contents. In this way, blueprint rendering becomes an effective visualization tool in applications, e.g., CAD systems, for interactively communicating structure and relationships of ancient architecture.

2. Blueprint Rendering

Blueprint rendering [ND04] extracts visible and non-visible edges of 3D models and then composes them in image-space. Thereby, visible edges are edges directly seen by the virtual camera and non-visible edges are edges that are occluded by faces of the 3D model, i.e., they are not directly seen. Technically, blueprint rendering combines depth peeling [Eve01] and an image-space edge enhancement algorithm [ND03] and can be implemented using hardware-acceleration [Kil04].

Layers of Unique Depth Complexity

Depth peeling decomposes arbitrary 3D models into disjunctive 2D layers of depth-sorted order, that we call *depth layers*. Generally speaking, depth peeling successively "peels away" layers of unique depth complexity.

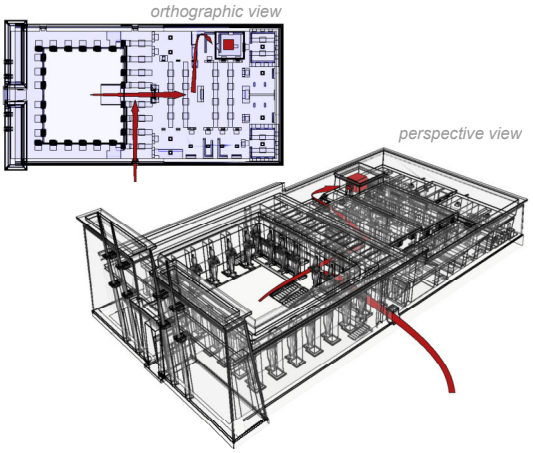

Figure 1: *Blueprints of perspective and orthographic views of the Temple of Ramses II in Abydos enhanced with glyphs for guidance.*

Commonly used real-time rendering generates the first depth layer. That is, the frame-buffer content contains pixels having a minimal z-value if an ordinary depth test has been performed. But, in this way, we cannot determine depth layers that come second (or third, etc.). Hence, depth peeling as a multipass rendering technique performs an additional depth test to extract those layers with respect to depth complexity. Based on depth peeling, we step deeply into a 3D model subject to the number of rendering passes while capturing the contents of each layer. Thus, we can extract the first *n* layers by *n* rendering passes (Fig. 2).

Performing Two Depth Tests

In the first rendering pass we perform an ordinary depth test resulting in the first depth layer and then capture the z-buffer and color buffer for further use. In consecutive ren-

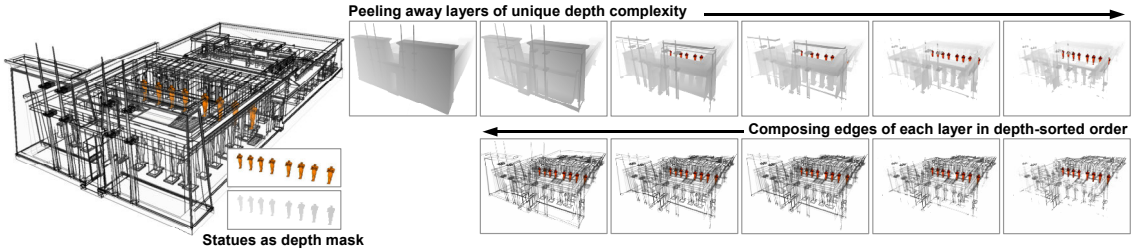

Figure 2: *Reducing the structural complexity in blueprints by considering a minimal number of depth layers with respect to a depth mask.*

dering passes we perform two depth tests. For it, we reuse the z-buffer of the previous rendering pass. We first test if a fragment is greater than the z-value of the targeted pixel location of the previous depth layer. If so, we, secondly, perform the ordinary depth test again. Otherwise, we reject that fragment prior. Finally, the frame-buffer content forms the next depth layer.

Extracting Visible and Non-Visible Edges

The edge enhancement algorithm extracts discontinuities in the normal buffer[1] and z-buffer as intensity values that constitute visible edges of 3D models. The algorithm preserves these edges as *edge map* for further use.

Combining both, depth peeling and edge enhancement allows us to extract edges for each depth layer. For it, we render encoded normal values instead of color values into the color buffer for a depth layer and, thus, can extract visible edges in each rendering pass because the z-buffer is already available. Furthermore, non-visible edges become visible when depth layers are peeled away successively. So, as a result, we can preserve visible and non-visible edges of 3D models.

Composing Blueprints

Finally, we compose blueprints using visible and non-visible edges stored in edge maps in depth-sorted order. We render each edge map as depth sprites into the frame-buffer. Thereby, we use color blending using edge intensities as blending factors and, e.g., a bluish, mixing color for providing depth complexity cues while keeping edges enhanced (Fig. 3).

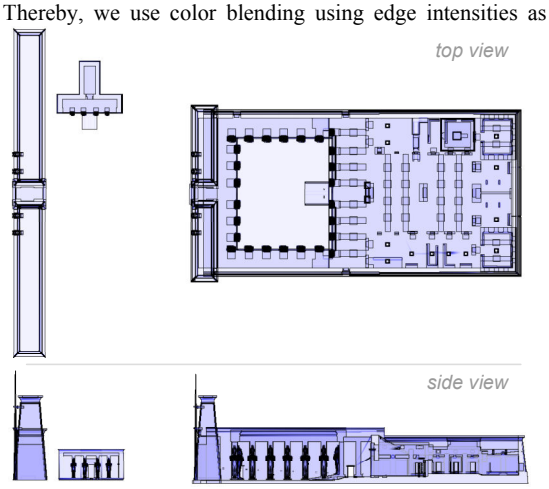

Figure 3: *Top and side views illustrate the layout of the temple.*

[1] A normal buffer contains geometrical normals of a 3D model encoded as color values.

3. Highlighting Hidden Components in Blueprints

We introduce depth masking to peel away a minimal number of depth layers until a specified fraction of occluded components of the 3D model become visible. For it, we construct a depth texture as *depth mask* of these components and render it as depth sprite in each rendering pass. In this way, we peel away depth layers until a specified fraction of these components become visible. Finally, we integrate and highlight these components when composing blueprints. In general, depth masking dynamically adapts the number of rendering passes and reduces the visual complexity of blueprints if the structural complexity of 3D models is excessive (Fig. 2).

4. Plan Views for Architectural Drafts

Blueprints generate plan views automatically to outline architecture comprehensible. Composing plan views using an orthographic camera for rendering is a straightforward task. Edges and depth complexity cueing are suitable to differentiate single components in an overall composition. In the plan views of the temple (Fig. 3), we can identify chambers, pillars, and statues systematically. Thus, blueprints increase visual perception in these drafts.

5. Relations and Locations in Architectural Depictions

We enhance blueprints of architecture with glyphs to communicate hidden details, locations, and relations. That is, we include general 3D geometry in blueprints to provide additional knowledge in our depictions of architecture. The illustrations in Fig. 1 mark a hidden chamber (red box) in the rear part of the temple and the pathways to it (red arrows).

References

[Eve01] Everitt, C., *Interactive Order-Independent Transparency*. Technical report. NVIDIA Corporation. 2001

[ND03] Nienhaus, M. and Döllner, J.: *Edge Enhancement – An Algorithm for Real-Time Non-Photorealistic Rendering*. Journal of WSCG'03, pp. 346-353. 2003

[ND04] Nienhaus, M. and Döllner, J.: *Blueprints – Illustrating Architecture and Technical Parts using Hardware-Accelerated Non-Photorealistic Rendering*. Proceedings of Graphics Interface 2004, pp. 49-56. 2004.

[Kil04] Kilgard, M. (Ed.): *NVIDIA OpenGL Extension Specifications*. NVIDIA Corporation. 2004.